# IN THE UNITED STATES PATENT AND TRADEMARK OFFICE

Utility Patent Application of

Donald H. McNeil

For

# ULTRA-MODULAR PROCESSOR IN LATTICE TOPOLOGY



Doc No 303-01

# ULTRA-MODULAR PROCESSOR IN LATTICE TOPOLOGY

RELATED APPLICATIONS:

US Provisional Application, serial number 60/277,745, filed March 22, 2001, and titled Information Management Software and Modular Processor Unit.

FEDERALLY SPONSORED RESEARCH:

Not applicable.

MICROFICHE APPENDIX:

Not applicable.

BACKGROUND OF THE INVENTION:

This invention relates to micro-computing systems and the management of miniature self-contained processors assembled in arrays. The invention further relates to such miniature processor arrays or lattices with the capability of prevailing format files-and-folders interface to host equipment such as personal computers, personal digital assistants (PDA), global positioning systems (GPS), digital cameras, mobile telephones, appliances, instruments, vehicles, etc.

The use of multiple processors, prior to their miniaturization, has taken various paths. Logue, et al., in US Pat. No. 4,268,908, show a modular microprocessor system with plural programmed logic arrays connected to a bus system for macro-processing. Each logic array executes a specific instruction beyond the standard set of instructions.

Hailpern, et al., in US Pat. No. 4,881,164, use a plurality of microprocessors assembled in an array with each controlling a respective area of a large memory. Barker, et al., in US Pat. No. 5,842,031 also use memory element processor arrays. These concepts have been expanded into single instruction multiple data (SIMD) architecture by Dieffenderfer, et al., in US Pat. No. 5,822,608 and Meeker, et al., in US Pat. No. 6,067,609.

With the advent of smaller non-volatile memory, such as non-volatile ROM and EEprom memory cells, pluralities of individual memory cells have been arranged in blocks to constitute an array. The circuit shown by Takashima, in US Pat. No. 5,903,492, uses a microprocessor to perform processing; and has an input/ output device with data storage connected to the microprocessor. The microprocessor is also connected to a semiconductor memory device including a plurality of memory cells each having a transistor gate and a ferroelectric capacitor.

Wallace, et al., in US Pat. No. 5,867,417, uses computer memory cards densely packed with a large number of flash EEprom integrated circuit chips thereon. The Wallace computer memory system provides for the ability to removably connect one or more of such EEprom carrying memory cards, to a host computer system through a common controller circuit that interfaces between the memory cards and a standard computer bus. Wallace also can provide each card with its own individual controller circuitry, which then makes it connectable directly to the host computer system's standard bus, without the need for a common controller circuit interface unit.

Microprocessors have been assembled in lattice structures, and other such arrays, by Klingman, in US Pat. No. 6,021,453, which shows an indefinitely extensible processor chain with self-propagation of code and data from the host computer end of the chain. Klingman has assembled a general purpose microcomputer with an "upstream" bus and a "downstream" bus. Klingman's upstream bus interfaces an integrated multiport RAM that is shared between an upstream processor and a local (downstream) processor. Local (downstream) interrupts are associated with dedicated locations in RAM. Klingman proposes arrays of such processors under the control of the host computer, wherein an indefinitely long chain of such processors can be utilized by one host computer.

Rohlman, et al., US application publication No. 20010032307, shows a microinstruction queue for an instruction pipeline within a microprocessor system. The pipeline has a plurality of units each with certain processing capabilities. At least one of the pipeline processing units can receive instructions from another pipeline processing unit, store the instructions and reissue at least some of the instructions after a "stall" occurs in the instruction pipeline.

Further, Nakano, in US Pat. No. 6,021,511, shows a processor with a plurality of execution units integrated into a single chip. The execution unit has an initial failure signal output device and a separate operating failure detection device. These devices each provide a respective failure signal in the presence of such failure in that unit. Failure signals are monitored by an allocation controller, which allocates instructions only between non-failed units.

Clery, in US Pat. No. 6,079,008, shows a parallel processing system processor with a plurality of execution units to repeatedly distribute instruction streams within the processor via corresponding buses. Clery uses a series of processing units to access his buses and to selectively execute his distributed instruction streams. His processing units individually may select and execute any instruction stream placed on a corresponding bus. These processing units autonomously execute conditional instructions, e.g., IF/ENDIF instructions, conditional looping instructions, etc. An enable flag within a processing unit is utilized to indicate the occurrence of conditions specified within a conditional instruction, and also to control the selective execution of instructions. An enable stack is utilized in the processing and execution of nested instructions.

These prior devices and systems, however, utilize multiple processors for such technical reasons as to increase the throughput of a centralized computing facility under control of system hardware and software. As will be demonstrated below, it is the object

of the present invention rather to provide highly independent, self-contained, and concurrent processing in the form of a peripheral attachment to host devices in a miniature package while presenting to any host having a compatible external bus an image of a passive mass storage volume, configuration of such devices to be under control of end users.

Such passive memory devices include compact flash (CF) and other such passive memory devices, which are connected to host system buses through CF card adaptors, such as shown by Yotsutani in US Pat. No. 6,109,931 and PCMCIA adaptors, such as shown by Moshayedi in US Pat. No. 5,660,568. These CF devices have been connected individually such as shown by Tsai in US Pat. No. 6,009,496, and into flash EEPROM memory system arrays. These arrays of passive memory devices have been connected to memory addressing controllers such as those shown by Harari et al, US application publication No. 2001/0026472 A1 and US 2001/0002174 A1 or to controllers such as shown by Tobita in US Pat. No. 6,275,436 B1. At times block memory addressing has been used as shown by Shinohara in US Pat. No. 5,905,993,

These CF and other memory devices are completely passive, being without any active computing element (CPU), which CPU is capable of supporting an operating system or executing application programs within a self-contained miniature module. While as stated above, the prior art has contemplated distributed intelligence, it has not contemplated distributed intelligence masquerading as passive memory.

A second object of the present invention is to provide a miniature modular computing device for performing independent and concurrent computations and input-output operations when attached to a host.

A third object of this invention is to provide this computing device with its own proprietary software and a self-contained operating system that enables it to operate as an intelligent media device, while presenting a passive virtual storage image to a host.

A further object of this invention is to provide this computing device where the projected look of a passive memory module effectively shares a virtual mass memory storage space within the device, while multiprogamming from that shared memory space concurrently with it being accessed from outside by a host.

A second further object of this invention is to provide lattice architecture for running proprietary operating system software permitting the interconnection of a plurality of these computing devices to operate as ultra-modular parallel processing units in relationship to one another and to have standardized interconnection hardware and protocol to a host.

A third further object of this invention is to provide a secure and reliable means for packaging, delivering, and running proprietary software applications in a self-contained subminiature module complete with a computing element (CPU), I/O circuits, an operating system, and application program(s).

An even further object of this invention is to provide the proprietary operating system software for the lattice network of plural computing device with the ability to configure the computing devices in scalable and dynamically end-user re-configurable combinations of units forming a self-contained parallel processing system.

SUMMARY OF THE INVENTION:

The objects of the present invention are realized in miniaturized self-contained computing system herein called a Modular Operating Topology Element (MOTE) imbedded in package similar in size to a Compact Flash™ (CF) unit or a PC Card, SmartMedia™ Memory Card (SMMC), Multi-Media Card™ (MMC) or other such

package. Further, the objectives of the present invention are realized in an array of such miniaturized self-contained computing systems and a method for logically interconnecting and managing the operation of such array as a smart lattice network.

Each MOTE, as for example, a 50 pin CF-sized package, has imbedded within it a programmable processor (CPU) with operating capabilities at least equivalent in memory addressing and interrupt handling facility to a Motorola® 68xxx processor. An internal hardware bus is connected to the CPU. Non-volatile random access memory (RAM) is connected to the hardware bus and provides working memory and mass storage memory for the device. Non-volatile read only memory (ROM) is also connected to the internal hardware bus and contains the dedicated operating system for the CPU, the software drivers for I/O attached to the device, and (optionally) application program code. A battery-backed real time clock-calendar unit is connected to the CPU through the hardware bus. Exception control circuits attached to the internal hardware bus provide for program-settable interval timer interrupts and watchdog timer interrupts to the CPU. Host interface I/O circuits connect the internal hardware bus to an external bus port for communication with an external computing system (host) hardware bus, e.g., PCMCIA bus, CF™ bus, SmartMedia™ bus, MMC™ bus, USB bus or other. The MOTE device may also optionally include I/O interface circuits for connection to one or more peripheral devices external to itself, e.g., for LED indicator lamps, for manual switches, or for serial, parallel or USB I/O, or other.

The device internal RAM module has within it a MOTE software-controlled allocation of both workspace memory for the MOTE's internal processing and a Virtual Mass Storage Control (VMSC) region, both of which are implemented under the control of the MOTE CPU and its operating system. All access to memory within a MOTE is via MOTE software which emulates a passive mass storage volume interface upon the host

bus, i.e., a host is presented with an image of the MOTE device as a virtual passive mass storage volume (in prevailing standard files-and-folders format, e.g., MS-DOS FAT 16, etc.) which it can only access under control of the interface emulation software within the MOTE. All application-level interactions between a host and a MOTE are conducted at the files-and-folders level, i.e., by the reading and writing of files in the VMSC which is logically shared by the host and the MOTE under the strict and exclusive control of the MOTE software.

Process management software within a MOTE is implemented in the form of a (non-Windows®, non-MacOS®, and non-UNIX) proprietary operating system to provide the services required to manage the MOTE CPU and the applications which run thereupon. CPU applications software may be written in any compatible language, such as the C or C++ languages. The operating system within each MOTE manages multiple logically concurrent logical processes as a circulating logical bus, with priority queues and unique addressing for each specialized logical process internal to it. Among the logical processes are those dedicated to MOTE operating system internal processing, to host bus management, to scheduling application events using the internal real-time clock-calendar circuits, to MOTE application software, to handling exceptions such as power-fail and warmstart, and to management of (optional) external port peripheral device communications.

A plurality of MOTE devices may be assembled into a host software-controlled lattice topology architecture, in which the MOTE internal soft-latticed topology becomes a system element or plural system elements for a larger host system wherein MOTE devices attached in parallel to a standard hardware external bus, e.g., CF or USB, receive messages in files written into their respective VMSCs by the host, then process those messages and return the results in files they write into their own respective

VMSCs. Implementation of the soft-latticed topology between a plurality of MOTE units connected within the lattice topology is thus effected by the operation in a host of a circulating logical bus algorithm similar to that of a MOTE. The process management of the lattice topology system emulates the fundamental ultra-modular multiprocessing algorithm implemented within a MOTE, but on a hardware external bus so as to implement the lattice topology at the system level. At this level, the algorithm addresses individual MOTE processing unit operations within the lattice topology, as opposed to logical process functions within a MOTE environment.

The invention permits logically concurrent processing at the MOTE level and physically concurrent processing at the system lattice topology levels. Process management at the system lattice topology level may be handled by a MOTE dedicated to that function and acting as a host if no other form of host is available. Re-configuration and re-sizing of systems of MOTE devices in a system level lattice topology are done at will by an end user by "hot-swapping" idle MOTE units, and the operating systems of a host and one or more MOTEs respectively automatically recognize such changes in subsequent operation. In particular, the operating system within a MOTE receives a power-fail interrupt when the MOTE is unplugged from an external host bus and saves the internal status of its operations so that it can automatically resume operation when it is plugged in again to a host external bus.

BRIEF DESCRIPTION OF THE DRAWINGS:

The features, advantages and operation of the present invention will become readily apparent and further understood from a reading of the following detailed description in connection with the accompanying drawings, in which like numerals refer to like elements, and in which:

Fig. 1 is a functional block diagram of a modular operation topology element (MOTE);

Fig. 2 is a functional block diagram of plural MOTE units connected to a host computer system through a standard external bus connection such as PCMCIA, CF, USB and others;

Fig. 3 is a block diagram for internal MOTE management (internal control program) operating under a circulating software bus algorithm;

Fig. 4 is a logic flow diagram for MOTE internal program flow:

Fig. 5 is a logic flow diagram for management by the MOTE operating system of host bus access to shared memory; and

Fig. 6 is a block diagram for a lattice topology of plural MOTE units operating under a re-circulating logical (software) bus algorithm.

Fig. 7 shows examples of four of the form factors suitable for MOTE packaging.

Fig. 8 shows two examples of hubs which accommodate multiple physical MOTEs in parallel.

Fig. 9 shows two examples of adaptors which can be used with MOTEs in order to accommodate them to alternative host buses.

DETAILED DESCRIPTION OF THE INVENTION:

The present invention provides a modular computing device herein called a Modular Operating Topology Element (MOTE) for performing independent computations and/or input-output (I/O) functions to operate as an attachment through a standard interface to a host system. A MOTE resides within a miniaturized package and can be programmed to function as an intelligent media device, a specialized processing device, a smart controller for peripheral input/output, or any other computer function whatsoever. A logically parallel software scheme is used to interconnect a plurality of

such miniaturized packages into a network having a lattice topology. Application software resides in one or more MOTE devices operating in parallel and concurrently. Host accessible externally distributed mass storage of data and computing resources can be distributed among a plurality of MOTE packages. All access by a host to the application functionality of a MOTE is through the reading and writing of files in the Virtual Mass Storage Control (VMSC) of the MOTE. In addition to non-volatile memory for mass storage, each package includes a self-contained processor (CPU), a dedicated operating system, and application software. This latter structure operates in conjunction with device internal memory to project a passive ("dumb") virtual mass storage logically formatted as files-and-folders in a prevailing standard format seen by any externally connected host system.

The physical architecture of a software-latticed network of MOTE units is re-configurable and re-scaleable externally at will by end users simply by plugging in and unplugging the units to a host external bus, e.g., PCMCIA or CF or USB, and configurations are automatically recognized internally by software within the host and the MOTEs respectively. The processing functions of MOTE may be changed by programming. In the absence of any other kind of host, one MOTE within the latticed network may perform the functions of the host bus operations controller. Other MOTE devices may be programmed to serve as specialized applications processors and as peripheral device controllers. Address queuing for each MOTE is established within the lattice. The operations controller implements ultra-modular and concurrent processing operations within the lattice architecture of a plurality of MOTEs. Data are made available to each MOTE as logical messages in files written to the MOTE's Virtual Mass Storage Control (VMSC), and results produced by the MOTE, including messages to be forwarded to other MOTEs, are returned in files which it creates on its own VMSC. A

non-hierarchical ladder interconnection topology can be implemented with re-circulating message exchange protocol.

As each modular computing device operates independently, it has the ability to interact with a host asynchronously, with the independent storage of "files" and "folders" data via the Virtual Mass Storage Control (VMSC) protocol resident within each unit. VMSC appears on any host external bus as a passive mass storage volume containing files and folders in, e.g., a prevailing standard format, such as MS-DOS FAT 16 etc. Generally, speed of interaction between a host and a MOTE is determined by the following factors: host speed, host software speed, bus speeds, MOTE circuit speeds, and MOTE software speed.

Each MOTE acts as a modular operating topology element to provide a convenient, small and standardized physical package, with full and immediate high-level compatibility with existing hardware and software host devices, such as desktop computers, laptop computers, personal digital assistants (PDA), global positioning systems (GPS), digital cameras, mobile telephones, appliances, instruments, vehicles, and any other devices which can support a standard external bus interface to virtual mass storage volumes. Each MOTE projects itself via VMSC as a virtual passive storage volume and is available for off-loading and parallel concurrent operation of application programs with exchange of data with a host system. Because of its own imbedded operating system and dedicated ROM-stored applications software, each MOTE is highly independent from the host operating system and from all of the software and hardware of any connected host except for the host external hardware bus and the host software drivers for that bus, both of which already meet prevailing external standards. With the soft-lattice architecture and its operations control, MOTE internal hardware and software, including management of the MOTE's internal soft-lattice

architecture and its internal control of software processes, is entirely invisible to and inaccessible by any host or other external agent.

The utilization of such modular computing device (MOTE) provides a cost-effective computing structure with extensibility of computing functions and of other capacities at will by end users. This extensibility is implemented simply by substituting or adding or removing MOTEs, which is possible as they are each standardized, interchangeable, self-contained units. Each MOTE being suited for modularity can therefore be easily assembled in arrays connected to standardized external bus connections (both connectors and connection protocol).

Each MOTE incorporates a computing element (CPU) and related circuits having functionality comparable to a Motorola® 68xxx unit. The simplicity of the logically concurrent processing implemented in MOTE internal software lends to efficiency and reliability without need for the burdensome complications of large, monolithic software operating systems such as Windows® or MacOS® or UNIX. The duplication within each MOTE device of computer hardware together with operating and application software, the highly independent operation of MOTE units, the complete inaccessibility to tampering from outside with the MOTE internal hardware and software, plus the abilities to be re-configured at will, to automatically restart after any power failure, and to recover from a variety of internal failures lends to an ultra-reliability of a MOTE deployed alone or in a soft-latticed network. The possibility of retrospective examination of data in VMSC files in a MOTE, including any such log files as a MOTE application may keep, also lends to ease of troubleshooting.

From a functionality standpoint, an individual MOTE unit, packaged for example with the form factor of a compact flash (CF) sized unit, is customized by the application software delivered within it. This software enables a MOTE to perform as a highly

independent unit to deliver, to carry, and to execute proprietary software and/or other downloaded software, operating fully in parallel and concurrently with a host and/or other MOTE units. Having received an input and associated data in files written by the host into its Virtual Mass Storage Control (VMSC), which appears to the host as a passive mass storage volume in a prevailing standard files-and-folders format, the MOTE performs the functions of the application software then resident within it. It then provides a response into one or more files in its VMSC without requiring intervention by any host specific software. MOTE operations are independent of host processing time and are carried out without exposing the MOTE resident proprietary software to the host in any way whatsoever. For extra security and privacy, a data encryption algorithm may be used to conceal VMSC resident proprietary information. Because the only access by the host to a MOTE is through files in a MOTE's VMSC, which is under strict control of the MOTE internal software, and because the MOTE cannot access the host except by posting files to it in its won VMSC, both host and MOTE are highly isolated from one another, leading to greatly enhanced reliability as well as to effective partitioning of functionality.

Because a MOTE may optionally have one or more external I/O interfaces other than that to the host external bus, it may with proper internal applications programming serve as a highly independent and concurrent peripheral processor to perform I/O functions to and from peripheral devices, e.g., through a serial port cabled to legacy devices or through a USB port to contemporary devices. In such an application, a MOTE can transfer data from files written to its VMSC by a host out through a peripheral interface and can store into files in VMSC any data received from that external interface, thereby making received data available for the host to read. Software in the MOTE application handles any necessary protocols and reformatting, performs any necessary

transformations on transmitted data, and effects error recovery for data transmission to and from the outside world.

For configurations of MOTE devices where there is no other host connection, a MOTE programmed as a host bus master controller is placed on the external bus. The MOTE host bus controller then implements VMSC requests to other MOTEs as needed, and other MOTEs in the configuration are behave as if they were attached to any other kind of host.

Applications software written for and delivered in a MOTE may implement any algorithm and access any I/O interface (other than direct access to the host external bus) which the hardware of the respective MOTE supports. To facilitate the writing of applications for use within the MOTE under its internal operating system, at least the services listed below are provided by that operating system via calls for the programming language being used, e.g., the C language. The following calls for service can be made to the MOTE process manager by an application:

| | |
|---|---|
| USE | - reserve and open a file in VMSC |
| DROP | - chose and release a file in VMSC |
| CREATE | - initiate a new file |
| DELETE | - remove a file from VMSC |
| COPY | - make a copy in VMSC |
| GET | - get the next record of a file in VMSC |
| PUT | - put a record into a file in VMSC |
| POINT | - position to specified data in a file in VMSC |
| READ | - read bytes directly from a file in VMSC |
| WRITE | - write bytes directly into a file in VMSC |
| SEARCH | - find data by content in a file in VMSC |
| PULL | - obtain data from the host bus through the MOTE synchronous conversational interface |
| PLACE | - make data available to the host bus through the MOTE synchronous conversational interface |
| SEND | - send a message to the logical bus within the MOTE |
| TAKE | - obtain the next message from the queue of the logical process |
| SENDOUT | - send a message to the logical bus within the MOTE |
| TAKEIN | - obtain the next message from the input queue from the host |
| ALLOCATE | - obtain a quantity of workspace in RAM |
| FREE | - return a quantity of previously allocated memory |
| RESERVE | - claim a resource |

| | |
|---|---|
| RELEASE | - free a reserved resource |
| START | - establish a new logical process in the MOTE |
| STOP | - remove a logical process from the MOTE |
| SETTIME | - set the MOTE real time clock |
| SETDATE | - set the MOTE real time calendar |
| TIME | - obtain current time |
| DATE | - obtain current date |
| SERIAL | - obtain MOTE serial number |
| LOG | - make an entry in a journal file in VMSC |
| WHEN | - establish the action to be taken when an exceptional event occurs |

Each MOTE 11, Fig. 1, contains a central processing unit (CPU) 13 having processing capabilities comparable to a Motorola® 68xxx processor. An internal hardware bus 15 connects the CPU 13 to an internal non-volatile RAM 17, being at least 1 megabyte in total size and directly addressable only by CPU 13. (This non-volatile RAM space is partitioned by MOTE software into a workspace portion 17a exclusively for its own internal use and a Virtual Mass Storage Control (VMSC) portion 17b where files and folders of the virtual storage volume image presented to the host under strict control of the MOTE operating software are managed.) A non-volatile ROM 19, addressable only by CPU 13, is also connected to the hardware bus 15. This ROM is also at least 1 megabyte in size and is used to hold MOTE operating system software and application-specific software.

Providing the CPU 13 with greater or lesser processing capabilities, providing the bus 15 at a different speed or byte width, and providing the RAM 17 and ROM 19 of larger or of smaller size will each to some degree impact upon the operational capabilities and the speed or throughput of a MOTE 11. These variations will not, however, alter the intent and function of a MOTE unit.

A battery-backed real time clock-calendar circuit 21 provides date and time to the CPU 13 via the hardware bus 15. A number (single or plurality) of input/output (I/O) devices 23, each having standardized interface hardware (connectors) and protocol may

optionally be connected to the hardware bus 15 of any particular MOTE. These I/O devices 23 are structured to meet prevailing hardware interface specifications and permit the CPU 13 to communicate with peripheral devices and ports such as to an LED indicator, a manual switch, a serial I/O port, a parallel I/O port, a USB port, and others. An interrupt control circuit 25 connected to the hardware bus 15, monitors for power failure, power resumption, watchdog timeouts, bus faults, timeslice interrupts, I/O interrupts, and other defined event interruptions, signaling such interrupts to the CPU 13.

A host bus external I/O circuit 27 connects the MOTE internal hardware bus 15 to the external physical and electrical interface of a connectable external host bus 28 through a standard interface such as PCMCIA, CF™, SmartMedia™, MMC™, USB or other. This circuit 27 provides the connectable host bus 28 with a way to present read or write addresses and data for transfer from and to the VMSC of the MOTE (implemented by software and physically resident in non-volatile RAM 17) under strict software control of the CPU 13 and its operating software. The MOTE software manages the electronic signals on this interface in such a way as to emulate the passive mass storage interface of a prevailing standard external memory unit (e.g., PCMCIA, CF™, SmartMedia™, MMC™, USB or other) having a capacity as seen by he host which is determined by the space allocated under MOTE software control to VMSC in the non-volatile memory 17. Every byte of data and every addressing assignment exchanged between the host and the VMSC is handled under strict control of the MOTE operating software.

A plurality of MOTEs 11a, 11b, etc., in Fig. 2 having identical hardware architecture (except for the number and kind of their optional I/O circuits) are connectable to a standard external host bus 29 connected into a host computer system

31. The host bus 29 meets prevailing standards for CF, PCMCIA, USB or other prevailing protocols. Each MOTE 11a, 11b, etc., is programmable to a specific application it is intended to carry on. Each MOTE 11a, 11b, etc., also projects virtual passive mass storage, i.e., VMSC, to the host 31.

The program management within each MOTE 11 unit, Fig. 1, is implemented under the MOTE's internal operating system, Fig. 3, which acts upon the internally programmed applications software present and active. These plurality of applications define a plural number of logical processes 33a - 33n, respectively, with any and all being operational at any one time period. These processes 33a - 33n are carried out in time-sliced, interrupt-preemptable multiprogramming within the MOTE which renders a MOTE as an internally ultra-modular and ultra-concurrent process manager implemented under software control. This ultra-modular and ultra-concurrent processing is carried out on input data messages from logical queues 35a - 35n, respectively, where these queues 35a-n are normally assigned and sequenced in order, unless that order is programmably reassigned for reasons of priority. A re-circulating software bus 37 is implemented under a software bus algorithm by software bus control 38. Messages sent as output to the software bus 37 from logical processes 33a - 33n are distributed by software bus control 38 to their respective destinations at logical queues 35a - 35n for further processing, for I/O, etc. Logical queues 35a - 35n for their respective logical processes 33a - 33n are maintained in non-volatile workspace memory by the software bus control 38 on a priority basis. In general, software bus control 38 manages logical processes on a time-sliced round-robin basis but preempts the active process when an asynchronous hardware interrupt from any source is received.

Noteworthy among the logical processes 33a - 33n is the Host Bus Service logical process 33a which manages all interaction between the host on the host external bus 39 and the Virtual Mass Storage Control (VMSC) 44 of the MOTE. Every byte of data read or written and every address presented to interface 39 by the host must pass through the strict control of this process 33a. Due to the activities taking place within the MOTE, the MOTE CPU (Fig. 1, 13) may not be able to pass control to the Host Service Logical Process immediately upon a request by the host interface 39 for service, so the "busy" indicator on that interface is set while uninterruptible internal processing is underway and reset when the MOTE is able to respond to bus interrupts.

Also noteworthy is the Clock-Calendar Timer Service logical process 33b which accepts messages from its logical input queue which request the reading or updating of the real time clock-calendar hardware registers 45 and also processes requests to schedule the sending of a notification message to another logical process when a particular time on a particular day occurs.

Further noteworthy, logical process such as 33i defined for an optional external port 40 is dedicated to and manages all I/O on that port.

MOTE 11 internal control program has its program flow logic illustrated in Fig. 4. When power to the MOTE unit 11 goes on 47, i.e., by the MOTE 11 being plugged into a host external bus, the hardware circuits are reset and interrupts are disabled 49 and appropriate data fields in the non-volatile memory are queried for a warm start pending state 51. If there is to be a warm start, then the previous status of all logical processes 33a - 33n and logical queues 35a - 35n and the software bus control 38 are restored 53. If there is not a warm start, then workspace, process queues and logical processes are initialized, step 55. Once step 53 or 55 is finished the next ready logical process is activated, step 57, and hardware interrupts (events) are enabled, step 58.

The event manager is then entered 60 to await the next interrupt of one of the following kinds: power failure interrupt 59; I/O interrupt 61, timeout interrupt 63, message posted to logical bus 65, real time clock interrupt 67, request to relinquish control 69, and malfunction or watchdog timeout 71. In the event of a power failure 59, the active process is suspended 73 and its status saved 75, then the system control data fields are set for a warm start 77 and the CPU operation is halted 79. The other interrupts 61 - 71 eventually all result in a return to step 57, the activation of the next ready logical process step.

In the event of an I/O interrupt 61, the active process is suspended 73, and a posting of the event appropriate to the process suspended is made, step 81. The logic then returns to step 57. If a timeout interrupt is received 63, it indicates that a process timeslice has expired, so the active process is suspended, step 73, and the logic then returns to step 57. When a message is posted to the logical bus 65, that message is distributed to the appropriate logical process queue 35a-35n in step 83, and the receiving process is readied to resume, step 85, and then the logic returns to step 57. A real time clock interrupt 67 causes the active process to be suspended 73, and a posting of the scheduled event to the appropriate process, step 81. A return to step 57 then occurs.

When a process relinquishes control voluntarily, step 69, it is suspended 73 and the logic returns to step 57, the activation of the next ready logical process step. When a watchdog timeout or other recognizable malfunction is detected, step 71, the offending process is aborted 87, the residual data are cleaned-up, step 89, and the logic returns to step 57.

Figure 5 illustrates in greater detail some of the functions of the Host Bus Service logical process 33a to show the logic flow for managing the host bus queries

carried on within a MOTE. A bus service interrupt is received, step 91, the "busy" indicator is set from the MOTE side 92, and the address presented by the host is checked for a valid VMSC location, step 93. If there is an error in the VMSC address, an addressing error is posted, step 95, and the process logic ends this routine 97. If a host requested address is valid as being within the VMSC area of RAM, then the address is accessed if it is within an active file for the host (or a file authorized for access by the host), step 97. If it is not, then an addressing error signal is posted, step 99. If the data address is valid, the data is transferred to or from that RAM address, step 101, the "bus :busy" indicator is reset, and the routine logic ends, step 97.

The implementation of the software lattice topology for a system of one or more MOTE units parallels the internal MOTE level algorithm. Figure 6 illustrates a block diagram for the lattice topology for plural MOTE units 100a – 100n operating as a host system network element under a re-circulating logical bus algorithm. In this host attachment topology there is a plurality of individually programmed MOTE processors 103a - 103n respectively. Each MOTE 103a-n has a respectively associated input queue file in its own Virtual Mass Storage Control (VMSC) 105a - 105n. Each MOTE 103a-n also has a respectively associated output queue files in its own VMSC 107a - 107n, respectively. A dedicated I/O device, 109a - 109n, may optionally be connected to any respective MOTE 103a-n. A host logical (software implemented) bus 111 under software control of the host 110 operates as a re-circulating information bus to make information available to each of the respective input queue files 105a-n, in turn for processing by the respective physical MOTEs 100a – 100n, and to collect information from each respective output queue file 107a-n, sequentially and distribute it as necessary. Host software bus control 110 is also responsible for recognizing the removal or addition of physical MOTE units 100a – 100n. (The physical connection of

each MOTE 100a – 100n to the host bus is not shown but is implied by the presence of VMSC access 105a – 105n and 107a 107n respectively; this connectivity is also shown schematically in the block diagram of Fig. 2.)

The form factors of some typical physical MOTE units are illustrated in Fig. 7. The Compact Flash™ form factor 121 is associated with an industry-standard 50 pin host bus interface 122. A PC card form factor 123 is associated with a 68 pin industry-standard PCMCIA host bus interface 124. A SmartMedia™ memory card form factor 125 is associated with an industry-standard 18 conductor host bus interface 126. A MultiMedia Card™ form factor 127 is associated with an industry-standard seven (7) conductor host bus interface 128. Each of these four interfaces is currently in use for passive mass storage cards in attachments with personal computers, laptop computers, personal digital assistants, digital cameras, appliances, instruments, vehicles, etc., and thus provides the physical and electronic bases for direct connection of a MOTE to a host (with host access to VMSC under MOTE software control) without modification to the host hardware or software.

Where several MOTEs are to be used together in parallel and concurrently, a hub device provides a base for their physical and electronic interconnection as shown in Fig. 8. A rectangular hub 131 or a circular hub 133 or other can be used. Such a hub provides supplementary power to the MOTE units 11a, 11b, etc., and an interface to an external host bus. If no host is connected to the hub, one of the MOTE units programmed as a host serves to control the host bus. In a hub, passive memory cards can be freely intermixed with MOTEs to provide additional mass storage capacity for a system. Consideration of a hub of MOTEs serves to highlight several advantages of the invention: external functional ultra-modularity in which can units can be assembled and

re-configured at will by an end user, and ultra-concurrency of MOTE units operating in parallel.

Where one or more MOTE units are to be physically and electronically attached to a host bus which is not directly compatible with the physical interfaces of the respective MOTEs, adapters may be used as shown in Fig. 9. For example, a MOTE having a CF host bus 147 may be plugged into a CF-compatible interface 145 of a CF-to-USB adapter 143 which may in turn be plugged into a host USB port via connector 141. Similarly, a MOTE having a CF host bus 157 may be plugged into a CF-compatible interface 155 of a CF-to-PCMCIA adapter 153 which may in turn be plugged into a host PC card port via connector 151. Industry-standard adapters are commercially available for CF-to-USB, SmartMedia-to-USB, MMC-to-USB, PCMCIA-to-USB, CF-to-PCMCIA, SmartMedia-to-PCMCIA, and others. In all of these configurations, the MOTE appears to the host as a passive mass storage volume due to the software within the MOTE and to the standard software presently available which manages the host side of the interface.

Many changes can be made in the above-described invention without departing from the intent and scope thereof. It is therefore intended that the above description be read in the illustrative sense and not in the limiting sense. Substitutions and changes can be made while still being with the scope of the appended claims.